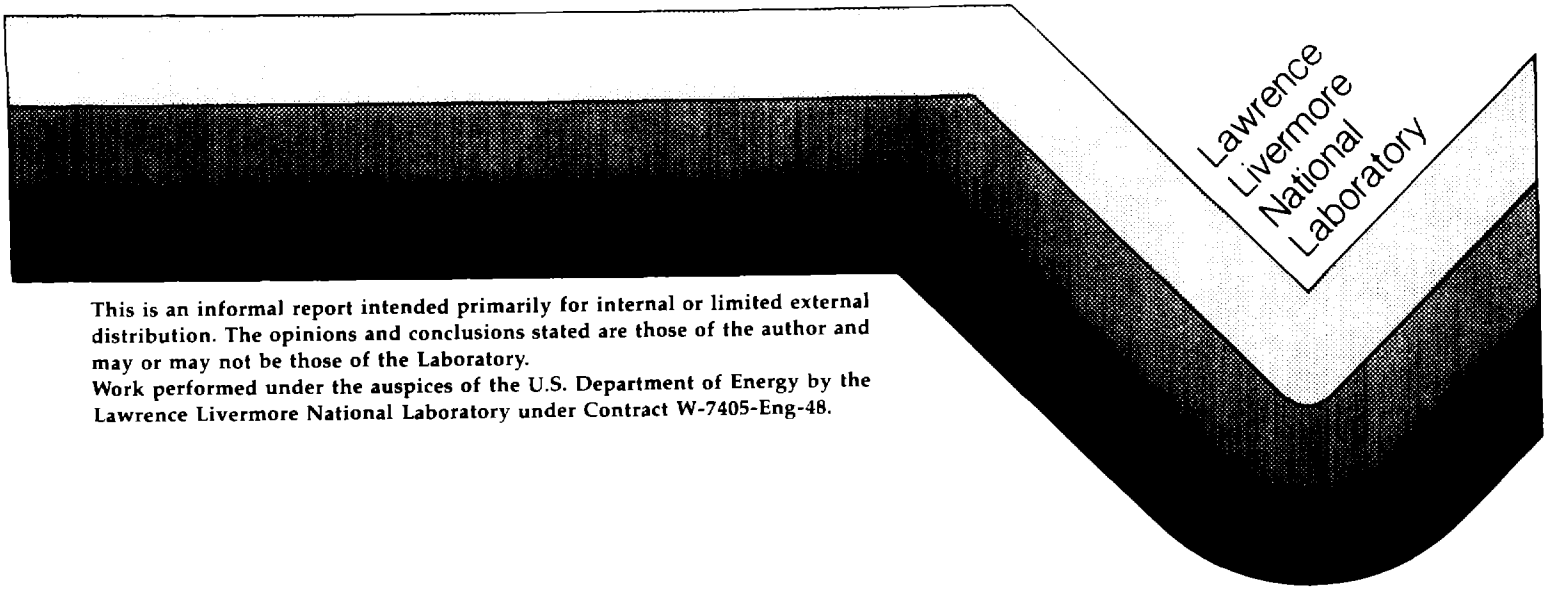
UCID- 20528

PRAXIS
RELEASE NOTES
VERSIONS 7.4 and 7.5

Frederick W. Holloway

September 2, 1985

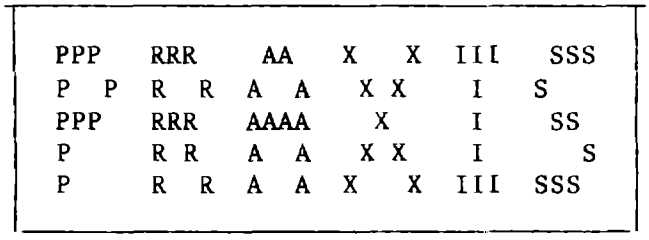


DISCLAIMER

Printed in the United States of America
Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161
Price: Printed Copy $; Microfiche $4.50

Page Range	Domestic Price	Page Range	Domestic Price
001-025	$ 7.00	326-350	$ 26.50
026-050	8.50	351-375	28.00
051-075	10.00	376-400	29.50
076-100	11.50	401-426	31.00
101-125	13.00	427-450	32.50
126-150	14.50	451-475	34.00
151-175	16.00	476-500	35.50
176-200	17.50	501-525	37.00
201-225	19.00	526-550	38.50
226-250	20.50	551-575	40.00
251-275	22.00	576-600	41.50
276-300	23.50	601-up[1]	
301-325	25.00		

[1]Add 1.50 for each additional 25 page increment, or portion thereof from 601 pages up.



Release Notes
Versions 7.4 and 7.5

Fred W. Holloway

Controls and Analysis Group
Nova High Energy Laser Facility
Laser Program
Lawrence Livermore National Laboratory
P. O. Box 5508, L-492
Livermore, CA 94550
(415) 422-5917

Generated on: August 30, 1985

file: disk_dev:[praxisdev.documents.version75]version75.doc

CONTENTS

Table of Contents

CHAPTER 4 SUGGESTED FUTURE IMPROVEMENTS

References

The microfiche of the VMS Symbolic Debugger, version 4.1

A complete list of references to the reports on Praxis is contained within the "Praxis Language Environment Distribution Package Description", UCID-30196 as revised January 1985.

Acknowledgment - Anthony De Groot

The addition of Symbolic Debugging support for Praxis programs under the VMS operating system was due completely to the efforts of Tony De Groot during the last six weeks or so prior to his leaving the Controls Group. This required major additions and modifications to the compiler, and substantial detective work in that the interface definition to the VSM Symbolic Debugger is not described or supported by DEC. The Symbolic Debugger should be a valuable tool for debugging of control system software. It has already proved valuable in debugging of the compiler itself.

CHAPTER 1

INTRODUCTION

1.1 PURPOSE OF THESE NOTES

These release notes are intended as a guide to those responsible for Nova software. They assume extensive knowledge of the present Praxis language.

Each improvement made in Praxis versions 7.4 and 7.5 is described. The descriptions are organized as shown in the table of contents. Many of the improvements have example programs which are contained within the chapter on example programs. For completeness, the final chapter lists the specific areas within the compiler which were modified. These will only be useful to those working on the compiler itself.

The improvements made in this version of the compiler include :

---------- Version 7.4 ----------

Source Line numbers passed through to the output files

Line numbers displayed on traceback

Support and interface for the VMS Symbolic Debugger

Support (partial) for the VMS Run Time Library

---------- Version 7.5 ----------

Integer Overflow detection

Improved support for Variadics

Fixed compiler crash on use of zero length structures

Corrected errors in declaration of PACKED structures

Improved run-time rangechecking

The Chapter on known bugs and suggested improvements was updated.

1.2 INSTRUCTIONS FOR USE

1.2.1 Global Symbols For Switching Between Versions

The PRAXIS_V74 and PRAXIS_V75 system-wide global symbol were added. All previous versions of the compiler have been removed from the system.

These symbols switch all logical symbols, symbols, and library names so that if a users command files are written to use the library name logical symbols as described above, a complete switch between the versions of the compiler can be accomplished by typing one of these symbols, then recompiling and linking with the normal command files.

PRAXIS_V74	- reassigns all logicals and symbols to point to version 7.4
PRAXIS_V75	- reassigns all logicals and symbols to point to version 7.5
PRAXIS_V	- print version number of Praxis now assigned to

CHAPTER 2

IMPROVEMENTS TO THE PRAXIS COMPILER

Unless otherwise stated, all changes apply to both the PDP-11 and VAX compiler. Most improvements have example programs that were used for testing, and the names of these examples are given.

2.1 LINE NUMBERS

2.1.1 Through To Compiler Output Files

During detail debugging, particularly at the software/hardware interface, the source line number which caused a particular set of executable instructions to be generated has always been needed. These line numbers are now passed completely through the compiler to the PDP-11 M11 files and the VAX macro listing files. These line numbers will also aid future efforts to optimize the compiler. The following pages show a sample source file listing and M11 listing from the PDP-11 compiler.

2.1.2 Lines That Have Only A <FF>

Lines that have only a <FF> now count as a line number. This allows a user to move down N records in a file to get to line N.

---------- Listing File----------

```
  1    //-------------------------------------------------
  2    // example of line numbers passed through to M11 file
  3    //-------------------------------------------------
  4    main module TESTDB
  5        declare
  6            AAA : integer
  7        enddeclare
  8
  9        for I := 1 to 10 do
 10            AAA := I
 11        endfor
 12        AAA := 4
 13    endmodule
 14
```

---------- M11 File----------

```
Title   TESTDB
psect   $Code1,rel,con,lcl,i,ro
1$1:
        jsr     pc,p$init
;                                               line number      6
;                                               line number      9
;                                               line number      9
;                                               line number      9
        tst     -(sp)
        mov     #1,-(sp)
1$3:
;                                               line number     10
        mov     @sp,2(sp)
        inc     @sp
;                                               line number     11
        cmp     @sp,#12
        ble     1$3
        tst     (sp)+
;                                               line number     12
        mov     #4,@sp
        tst     (sp)+
        rts     pc
end     1$1
```

2.1.3 Line Numbers Displayed On Traceback (VAX)

The Praxis compiler now puts line number information in the VAX object file so that if the program crashes, the traceback dump will show the line numbers on the call stack.

2.2 INTERFACE TO THE VAX VMS SYMBOLIC DEBUGGER

The Praxis compiler now supports many (but not all) of the features of the VMS Symbolic Debugger. The following features are supported:

Breakpoints at any line

Source line display

Single stepping by lines

Examination/modification of static variables by name

Register examination

Any debugger command that can be issued for macro code

The following feature is not supported:

Examination/modification of dynamic variables by name.

Note however, that dynamic variables can be accessed by reference to the generated macro code and use of the macro code debugger commands (i.e. register offsets, etc.).

2.2.1 Use_routine_names Default True On VAX

To support the use of the Symbolic Debugger on the VAX, the Use_routine_names compiler directive was changed to be true for the VAX compiler by default. The default value is still false for the PDP-11 compiler.

2.2.2 How To Use The VMS Symbolic Debugger With Praxis

All Praxis sources compiled with version 7.4 and subsequent versions

will have the necessary information for the debugger interface within their object files. To use the VMS Symbolic Debugger, the image should be linked with /debug. No loss in run time efficiency occurs if the program is not linked with the debugger. Users should study the Symbolic Debugger Manual.

Some important commands are:

set module/all	- Debugger reads in symbols for all modules. One should probably perform this command before doing anything else.
set break/line	- Set break point at each line. Useful for single stepping through code which has no matching Praxis source program. (cancel break/line - removes it)
set scope module_name	- Sets the scope of the debugger to the desired module. For many commands issued, the current module is assumed unless explicitly specified.
set break %line N	- Where N is the number of line from top (line 1) in the praxis source module. The listing file will also provide the desired line number.
set mode screen	- Screen mode debugging. More detail is given later.
examine/decimal V	- Examines static variable V in your program. Returns the value in decimal. Dynamic variables cannot be examined by name. They can be examined by looking at register contents and appropriate offsets. Looking at the code generated by the compiler will generally make it clear where the dynamic variables are stored.
examine/instruction @pc	- examines the instruction to which program counter is pointing. Can be used to figure out where dynamic variables are stored.
go	- Starts the program.
step	- Single steps the program.

Setting debugger to screen mode will show the program source in the upper half of the screen, and the outputs from the program and debugger commands in the lower half of the screen. An example of a screen mode debugging is shown on the following page. The program is suspended at line 25.

```
--SRC: module TESTLINE---source-scroll----------------------------------------
    19:     while a=1 do
    20:         b := 1
    21:         b := 2
    22:     endwhile
    23:
    24:     a := 1
->  25:     b := 2
    26:
    27: endmodule
--OUT---output----------------------------------------------------------------
    19:     while a=1 do
TESTLINE\PRXAJD\A:      0
TESTLINE\PRXAJD\B:      2
stepped to TESTLINE\PRXAJD\L$8
    22:     endwhile
stepped to TESTLINE\PRXAJD\%LINE 24
    24:     a := 1
stepped to TESTLINE\PRXAJD\%LINE 25
    25:     b := 2
------------------------------------------------------------------------------
```

2.3 RUN TIME LIBRARY IMPROVEMENTS

2.3.1 Screen Management Routines

The Screen Management Routines of the VMS Run Time Library were tested, and as a result various improvements to the stubs to the Run Time Library (RTLSTUBS.PRX) were made, and the module SMGDEF.PRX was added. SMGDEF should be "used" to obtain constants described within the Screen Management Routine descriptions in the VMS manuals.

An example routine was written (PASTE.PRX) which establishes four virtual status displays and one virtual command display on a VT-100 based "pasteboard." The command display is used to read commands. A number (1,2,3 or 4) is entered to select the status display to be changed. The Up, Down, Left, and Right cursor control keys can then be used to move the selected status display. The Gold key (PF1) can be used to toggle the mode between moving the selected display and changing its size.

Note that the command line editor must be turned off (Set Term/noline_edit) to run this example because it uses the Up and Down arrows.

2.4 VARIADIC ARGUMENT IMPROVEMENTS

2.4.1 Small Size Variadics

Prior to version 7.5, small size objects passed to a procedure which expected a variadic number of arguments were not accessed correctly by the called procedure. This has been fixed. See the example program VARADIC1.PRX

2.4.2 Zero Arguments Passed To Variadic Parameter

Prior to version 7.5, if no arguments were passed to a parameter that was declared to be variadic, the PDP-11 code generated would crash. This has been fixed. See example VARIADIC1.PRX.

2.4.3 Variadic Flexible Arrays

Support for Variadic Flexible array arguments was added during version 7.3 and improved in version 7.5. See example program VARIFLEX.PRX. It was discovered that in version 7.3 the flexible arrays were all pushed on the stack prior to calling the procedure. This would work (inefficiently) on the VAX but would crash the PDP due to lack of stack space. This has been fixed in version 7.5 so that only the address of the arrays are pushed on the stack.

2.4.4 Keyword References To Variadic Parameters

Objects passed to a procedure as a variadic argument may be referenced in the keyword style. Each reference is in order. See example program VARIKEY.PRX

2.5 OVERFLOW DETECTION

2.5.1 Integers And Cardinal

Integer and cardinal overflow detection has been implemented on the VAX and improved on the PDP within version 7.5 of Praxis. Integer overflow detection appears to work correctly in all situations where there are machine instructions to perform the desired operation on the desired size operands. For example, add, subtract, and multiply of 8 bit, 16 bit, and 32 bit operands on the VAX, and the same operations on 16 bit operands on the PDP. On the PDP, support for overflow detection of self-modified operands was made to work correctly (i.e. A *= +1). Other types of operations already worked on the PDP. See OVFLTEST.PRX as an example. Cardinal overflow detection also seems to work correctly, however, few actual tests were performed.

The compiler implements operations on sizes of operands which are not supported by the machine hardware as a series of machine instructions, often involving temporary variables, shifting, unpacking and packing operations. It would be quite difficult to implement complete overflow testing in situations not supported directly by machine hardware - 9 bit integers for example.

Further testing through usage may turn up situations where integer overflow detection is not performed correctly, since there are an infinet number of combinations to test. It is believed that any such problems could easily be fixed in the future provided they involve instructions for which the machine hardware provides single instruction support.

2.5.2 Floating Point Overflow

The detection of floating point overflow on the VAX is done directly by the hardware and results in a hardware fault trap instead of a bit being set in the program status word (PSW). It would be feasible in the future to write a procedure to catch the hardware trap and this could be done if it were worth while to users of the language.

2.6 PACKING

2.6.1 Packed Structures Of Unpacked Objects

Prior to version 7.5 packed structures with unpacked arrays of unpacked objects which were over 32 bits in size but not an even number of 16 bits in size were constructed incorrectly. The size of objects which are greater than 32 bits in size (16 bits in size on PDP) are supposed to be rounded up to the nearest 16

bits. This was not done in the routine that performs structure declarations in the compiler prior to version 7.5. See the example program ODDPACK.PRX.

2.7 RANGECHECKING IMPROVEMENTS

2.7.1 Rangechecking On Size Conversions At Run-time

Previous versions of the compiler performed incorrect run-time rangechecking when a size conversion was required between the source and the destination (for example, A := B, where A is 16 bit integer and B is 32 bit integer). This was corrected.

2.7.2 Rangechecking Of Expression Evaluation At Run-time.

Previous versions of the compiler would not range check the result of a source expression prior to the answer being transfered to the destination. (for example, A := B + 1). This was corrected.

2.7.3 Optimization Of Run-time Rangechecking.

Optimization was added to run-time rangechecking such that if it is obvious that the source cannot be larger than the destination run-time rangechecking is not performed. (for example, A := B when A has a declared range and size which completely incompases the declared range and size of B). This substantially reduced the amount of code generated when rangechecking is armed.

2.8 DEBUGGING IMPROVEMENTS

2.8.1 Better Diagnostic Messages

2.8.1.1 ADDRESSOF Function Argument Diagnostic Added -

The function ADDRESSOF cannot return the address of a constant. Previous releases of the compiler diagnose an attempt by the user to do this, and in fact generated bad code when it was attempted. The argument for ADDRESSOF must be a variable. If the argument for ADDRESSOF is not a variable, a diagnostic message "need to use variable here instead of <what was there>" is generated.

2.8.1.2 Better Explanation Of Some Diagnositics -

A bug was fixed within the internal routine print_mode to pass an explain flag to recursive calls. This should result in clearer diagnositics in some situtations.

2.8.1.3 Source Lines Over 132 Characters Diagnostic Message Added -

Source line over 132 characters long used to generate an abstract error message, leading to confusion about the real problem. A better diagnostic was added. Source lines larger than 132 characters long now generate the message "Record in source file bad, probably too long."

CHAPTER 3

EXAMPLE PROGRAMS

3.1 INTEGER OVERFLOW DETECTION

```
//-------------------------------------------------
//      Example of Integer Overflow detection
//-------------------------------------------------

main module OVFLTEST

        use TEXTIO

        declare
          C : static 32 bit integer
        enddeclare

        ARM X_OVERFLOW

        guard
            C := -1
            repeat
                C *= *2
            until false
        catch
            case x_overflow:
                out_line (TTY, "Overflow")
        endguard
endmodule
```

3.2 VARIADIC ARGUMENTS OF SMALL SIZE

```
//-----------------------------------------------------------------------
//      Example of the use of Variadic arguments
//      of small size and of zero quantity
//-----------------------------------------------------------------------
main module VARITEST1
    use TEXTIO

    procedure VARITEST1
        param
            Dummy       : in val integer
            Position    : variadic in val integer range 0..15   // BUG FIXED
                                                                // small value args
        endparam

        out_integer (TTy, High(Position), " Number of variadics = ")
        out_string  (TTY, "<HT>: ")

        for I := 1 to high (Position) do
            out_integer (TTY, Position [I], " ")
        endfor
        out_record  (TTY)

    endprocedure

    VARITEST1 (1,2,3,4,5,6,7)
    VARITEST1 (1,2,3,4,5,6)
    VARITEST1 (1,2,3,4,5)
    VARITEST1 (1,2,3,4)
    VARITEST1 (1,2,3)
    VARITEST1 (1,2)
    VARITEST1 (1)                       // BUG FIXED: # Variadics = 0

endmodule
```

3.3 VARIADIC NUMBER OF FLEXIBLE ARRAYS

```
//-----------------------------------------------------------------
//      Example of Use of Variadic number of flexible arrays
//-----------------------------------------------------------------
main module VARIFLEX

    use TEXTIO

    procedure VAR (a: variadic in ref packed array[1..?n] of char)

        // Note that high() works for both number of variadics and length
        // of flexible arrays

        for i := 1 to high(a) do
            out_string (tty, a[i], high(a[i]))
            out_string (tty, " ")
        endfor
        out_Record (tty)
    endprocedure

    VAR ("HI", "THERE", "YOU", "ALL")

endmodule
```

3.4 KEYWORD REFERENCES TO VARIADICS

```
main module VARIKEY
    use TEXTIO

    procedure VARIKEY   (AAA : variadic in val integer)
        for I := 1 to high (AAA) do
            out_integer (TTY, AAA [i])
            out_record  (TTY)
        endfor
    endprocedure

    VARIKEY (AAA:100, AAA:-200, AAA:300)
endmodule
```

3.5 PACKED STRUCTURES OF ODD SIZE UNPACKED OBJECTS

```
//-----------------------------------------------------------------------
//      Example of Packed structure of odd size unpacked objects
//      (crashed compilers prior to version 7.5)
//-----------------------------------------------------------------------

module ODDPACK
    export      Time_Base_Type
    declare
        Samp_inter_type is 8 bit integer
        Brpts_type is structure                 // 40 bits declared
            Loc         : integer
            Samp_inter  : Samp_inter_type initially 1
        endstructure

        Time_Base_Type is packed structure
            Brpts       : packed array [1..2] of brpts_type
            Samp_trig   : integer
        endstructure
    enddeclare
endmodule
```

3.6 USE OF RTL SCREEN MANAGEMENT ROUTINES

```
//----------------------------------------------------------------
// example of use of SMG routines of the RTL
//
// Note: the Command Line Editor must be disabled to run this
// (Set term/noline_edit)
//----------------------------------------------------------------
main module PASTE
    use TEXTIO
    use SMGDEF,SSDEF
    use RTLSTUBS
    use VMSSTUBS, VMSTYPES, VMSPROC

    declare

        N_Status_Displays = 4
        Selected_Display        : integer initially 1
        Operation               is [Move, Change_Size]
        Selected_Operation      : Operation initially Move
        Pasteboard_ID           : long_cardinal
        Keyboard_ID             : static long_cardinal
        Keytable_ID             : long_cardinal
        N_pasteboard_rows       : long_integer
        N_Pasteboard_Columns    : long_integer

        Status_Display_ID       : array [1..N_Status_Displays] of long_cardinal
        Command_Display_ID      : long_cardinal
        Display_Message         : descriptor
        Command_Message         : descriptor
        SMG_Status              : static long_cardinal
        Recieved_Text           : descriptor
        Recieved_text_buffer    : static packed array [1..120] of char
        Display_message_buffer  : packed array [1..9] of char
        Prompt_string           : array [Operation] of descriptor


        Position_Array is array [1..N_Status_Displays] of long_integer
        Status_Position_Row     : Position_Array initially Position_array (
                                  [1]:1, [2]:5, [3]: 10, [4]:15)
        Status_Position_Column  : Position_Array initially Position_Array (
                                  [1]:1, [2]:5, [3]: 10, [4]:15)
        Status_Size_Row         : Position_Array initially Position_array (
                                  [1]:5, [2]:5, [3]: 5, [4]:5)
        Status_Size_Column      : Position_Array initially Position_Array (
                                  [1]:20, [2]:20, [3]: 20, [4]:20)


        // special keys

        up_message              : static descriptor
```

```
        down_message            : static descriptor
        left_message            : static descriptor
        right_message           : static descriptor
        gold_message            : static descriptor

        up_key                  : static descriptor
        down_key                : static descriptor
        left_key                : static descriptor
        right_key               : static descriptor
        gold_key                : static descriptor

    enddeclare



    procedure Define_Special_Keys
        param
            Keytable_ID : in ref long_cardinal
            keyboard_ID : in ref long_cardinal
        endparam

        Load_string_descriptor (Up_Message, "UP")
        Load_string_descriptor (down_Message, "DOWN")
        Load_string_descriptor (left_Message, "LEFT")
        Load_string_descriptor (right_Message, "RIGHT")
        Load_string_descriptor (gold_message, "GOLD")

        Load_string_descriptor (Up_key, "UP")
        Load_string_descriptor (down_key, "DOWN")
        Load_string_descriptor (left_key, "LEFT")
        Load_string_descriptor (right_key, "RIGHT")
        Load_string_descriptor (Gold_key, "PF1")

        declare
            one_bit_logical is 1 bit logical initially 2#0
            Attrib_type is packed array [0..31] of one_bit_logical
            Attrib : Attrib_type initially Attrib_type (
                [SMG'$V_KEY_NOECHO]:    2#1,
                [SMG'$V_KEY_TERMINATE]: 2#1)
        enddeclare

        SMG_Status := SMG'$ADD_KEY_DEF (Keytable_ID, UP_Key,
                        attributes:Attrib,
                        equiv_string:UP_message)
        SMG_Status := SMG'$ADD_KEY_DEF (Keytable_ID, Down_Key,
                        attributes:Attrib,
                        equiv_string:Down_message)
        SMG_Status := SMG'$ADD_KEY_DEF (Keytable_ID, LEFT_key,
                        attributes:Attrib,
                        equiv string:Left_message)
        SMG_Status := SMG'$ADD_KEY_DEF (Keytable_ID, Right_Key,
                        attributes:Attrib,
                        equiv_string:Right_message)
```

```
        SMG_Status := SMG'$ADD_KEY_DEF (Keytable_ID, Gold_Key,
                        attributes:Attrib,
                        equiv_string:Gold_message)

        SMG_Status := SMG'$SET_KEYPAD_MODE (Keyboard_id, cardinal (0))
    endprocedure


        Display_message_Buffer := "DISPLAY N"

        Load_string_descriptor (display_message, Display_message_buffer)
        Load_string_descriptor (Recieved_text, Recieved_text_buffer)
        Load_string_descriptor (Prompt_string[Move], "Move which way?")
        Load_string_descriptor (Prompt_string [Change_Size], "Size which way?")


    SMG_Status := SMG'$CREATE_PASTEBOARD (Pasteboard_ID,
                        PB_rows:N_Pasteboard_rows,
                        PB_Columns: N_Pasteboard_Columns)


    for N := 1 to N_Status_Displays do

        SMG_Status := SMG'$CREATE_VIRTUAL_DISPLAY
                        (Status_Size_Row [N],
                         Status_Size_Column [N],
                         Status_Display_ID [N],
                         SMG'$M_BORDER, SMG'$M_REVERSE)

        SMG_Status := SMG'$PASTE_VIRTUAL_DISPLAY
                        (Status_Display_ID [N],
                         Pasteboard_ID,
                         Status_Position_Row [N],
                         Status_Position_Column [N])


        Display_Message_Buffer [9] := char (N + integer ($1) - 1)

        SMG_Status := SMG'$PUT_CHARS
                        (Status_Display_ID [N],
                        Display_Message, 1, 10)
    endfor


    SMG_Status := SMG'$CREATE_VIRTUAL_DISPLAY (1, 60, Command_Display_ID,
                        SMG'$M_BORDER)  ///, SMG'$M_REVERSE)


    SMG_Status := SMG'$PASTE_VIRTUAL_DISPLAY
                        (Command_Display_ID, Pasteboard_ID,
                                21, 2)

    SMG_Status := SMG'$CREATE_VIRTUAL_KEYBOARD (Keyboard_ID)
```

```
    SMG_Status := SMG'$CREATE_KEY_TABLE (Keytable_ID)

    Define_Special_Keys (Keytable_ID, Keyboard_ID)


repeat

    SMG_Status := SMG'$SET_CURSOR_ABS (Command_Display_ID, 1, 1)
    SMG_Status := SMG'$DELETE_LINE (Command_Display_ID, 1, 1)

    declare
        Recieved_string_length : word_cardinal
        Legal_Input : boolean initially false
    enddeclare

    SMG_Status := SMG'$READ_COMPOSED_LINE
                      (Keyboard_ID, Keytable_ID,
                      Recieved_Text, Prompt_String [Selected_Operation],
                      Recieved_string_length,
                          Display_ID:Command_Display_ID)

    select Recieved_text_buffer [1] from
        case $U:
            select Selected_Operation from
                case Move:
                    Status_Position_Row [Selected_Display] *= -1
                    Legal_Input := true
                case Change_Size:
                    Status_Size_Row [Selected_Display] *= -1
                    Legal_Input := true
            endselect
        case $D:        //      SMG'$K_TRM_DOWN:
            select Selected_Operation from
                case Move:
                    Status_Position_Row [Selected_Display] *= +1
                    Legal_Input := true
                case Change_Size:
                    Status_Size_Row [Selected_Display] *= +1
                    Legal_Input := true
            endselect

        case $L:        //      SMG'$K_TRM_LEFT:
            select Selected_Operation from
                case Move:
                   Status_Position_Column [Selected_Display] *= -1
                   Legal_Input := true
                case Change_Size:
                    Status_Size_Column [Selected_Display] *= -1
                    Legal_Input := true
            endselect

        case $R:        //      SMG'$K_TRM_right:
            select Selected_Operation from
                case Move:
```

```
                Status_Position_Column [Selected_Display] *= +1
                Legal_Input := true
                    // SMG_Status := SMG'$PUT_CHARS
                    // (Command_Display_ID, right_Message, 1, 1)
            case Change_Size:
                Status_Size_Column [Selected_Display] *= +1
                Legal_Input := true
        endselect

    case $G:
        if Selected_Operation = Change_size do
            Selected_Operation := Move
        otherwise
            Selected_Operation := Change_size
        endif
    case $1:
        Selected_Display := 1
    case $2:
        Selected_Display := 2
    case $3:
        Selected_Display := 3
    case $4:
        Selected_Display := 4
endselect

if Legal_Input do
    select Selected_Operation from
        case Move:
            SMG_STATUS := SMG'$MOVE_VIRTUAL_DISPLAY
                    (Status_Display_ID [Selected_Display],
                     Pasteboard_ID,
                     Status_Position_Row [Selected_Display],
                     Status_Position_Column [Selected_Display])
        case Change_Size:
            SMG_Status := SMG'$CHANGE_VIRTUAL_DISPLAY
                    (Status_Display_ID [Selected_Display],
                     Status_Size_Row [Selected_Display],
                     Status_Size_Column [Selected_Display],
                     SMG'$M_BORDER, SMG'$M_REVERSE)
    endselect
endif

until false


endmodule
```

CHAPTER 4

SUGGESTED FUTURE IMPROVEMENTS

4.1 DECLARATIONS

4.1.1 Packed Packed Structures Of Characters

Packed Packed declarations were in the language design but were not completely implemented, and have never been used on Nova. However, Packed Packed structures, when all the items of the structure are either characters or arrays of characters, appear to work correctly but should be tested thoroughly. They are very handy for use as organized I/O blocks. Other combinations of packed packed times should be tested to find out how much work it would be to implement.

4.1.2 Equivalent Integer And Cardinal Constants

Integer and Cardinal Constants (i.e., 1, 2, 5) should be equivalent so that it's not necessary to specify cardinal(2) every time one needs a 2 in an expression that requires a cardinal.

4.1.3 Silly Declarations

Some silly declarations should be disallowed. For example:

```
A is integer
B is array [A] of ...
```

Is legal, but shouldn't be (i.e., extremely big array).

4.2 DYNAMIC ALLOCATION ISSUES

4.2.1 Allocation Of Objects With Holes

The allocation of a flexible array of items with holes in them, has the same inherent problem as holes in other objects described above. However, for Version 7.2 no clearing of allocated items with holes was implemented.

4.2.2 Type Check Of Allocate Size Variable

There is no type check of the allocate size variable. See the number 0.4 in the following example - which compiles!

```
declare
    Flex_array      is packed array [1..?n] of char
    P               is pointer Flex_array
    T1:             static P
enddeclare

T1 := allocate P (0.4)
```

4.3 INITIAL CONDITIONS

There are still some examples of complicated combinations of structures and arrays and multiple imports that do not set initial conditions in an understood manner. This needs further study, but is quite useful now for all but the most complicated situations.

4.3.1 Initially Sizeof

The declartion sentence "C : cardinal initially sizeof (A)" returns zero regardless of size of A. See EVAL.PRX

4.4 SETS

The entire concept of the use of sets is not well understood within the present users of the language. Testing of the use of sets indicates that on the Vax compiler:

declarations work

initially works

constructors do not work

the <= and < operators fail in the syntax phase of compilation,

+, -, * operations don't work for sets of over 32 items.

Nothing regarding sets works on the PDP-11. Good useful example programs need to be prepared and tested. see SETTEST.PRX.

4.5 MATHEMATICAL OPERATIONS

4.5.1 Reincorporate The Conversion Routines

Pick up the differences in the routines used in Laser Diagnostics. Test the conversion routines - make a test package. LSI-11?

4.5.2 Unsigned Comparisons

Unsigned comparisons are not implemented. This means that comparisons of cardinals is done with signed comparisons. Therefore, if a cardinal is large enough to use what would be the sign bit of the size of the instructions available on the respective machines (8, 16, 32), the less-than or greater-than comparison will be incorrect. The equal or not-equal comparison is still correct. This effects both mathematical user written statements and range checking. Also, since the compiler itself is written in Praxis, compile-time range checking of 32 bit cardinals is not implemented.

4.5.3 Real Exponentiation

Exponentiation is not implemented for real bases. Therefore, i ** j is supported, but r ** j is not.

4.5.4 Long Real Constants

Long real constants on VAX is not supported. The following does not work:

```
r : long_real initially 1.0
```

Instead, one must type:

```
r : long_real initially long_real (1.0)
```

{This should be simple to fix} Also, suspect that other long real operations not correct.

4.5.5 Long Real On PDP Not Implemented.

4.5.6 Round, Floor, Ceiling For The PDP-11

Run time procedures that perform round(), floor(), and ceiling() need to be implemented within the runtime support package for the PDP-11. Compiler calls to the routines exist.

4.6 PROCEDURE INTERFACE ISSUES

4.6.1 Conversion Of Real To Integer On PDP

The conversion of real to integer on PDP does not work in the following example: (ie the CGTABLES pattern may be wrong)

```
procedure CONVERT (X:integer)
endprocedure

declare (A = 2000.0)

CONVERT (integer (A) )

// Note that the following works:

Declare (I : integer)
I := integer (A)
```

4.6.2 Status Of "module_name . Imported_from" Syntax?

4.6.3 Confusing N On Multiple Flexible Arrays

Parameter statement of

```
A,B : inout packed  array [1..?N] of char
```

is accepted, but which does N apply to?

4.6.4 OUT Flexible Array - What Is It?

Why is out flex array unimplemented? What are its rules?

4.6.5 Functions Which Return Flexible Arrays

Can be coded and compile without error on VAX, but don't run correctly. On PDP, compiler error "inconsistent stack address assignment." This should either be fixed, or a diagnostic should be inserted to prevent users from coding such a return. See FUNSTGTST.PRX for example.

4.6.6 Reference To Part Of Structure Returned From Function

Functions can return a structure, but a reference to a portion of the structure directly in the sentence with the function call will compile without error but will not run correctly. The function is never called. For example:

```
//-------------------------------------------------------------------
// PRAXIS BUG - never calls function TEST....
//-------------------------------------------------------------------
main module PARTFBUG
    declare
        A is structure
                J : array [1..10] of integer
                B : integer
                C : integer
        endstructure
        Joe : static A
    enddeclare

    function test ()
        returns test:A
    endfunction

    Joe . B := Test () . B              // This line has a bug in it....
endmodule

.ENDLITERAL
```

```
.HL1    Flow Control
.HL2    Tag field inside a Variant Structure
   Present compiler allows select on a tag field inside a variant - not good.

.HL1    Range Checking

.HL2    Range Checking of Different size objects
        Range checking is not performed correctly during the assignment of
a 16 bit integer to a 32 bit integer. The whole subject of range checking
between different size objects needs to be investigated.



.HL1    Efficiency Problems
```

.HL2 Large Range of Cases

Select on an integer with a very large range of cases (1, 100000) causes
the compiler to take an N*N amount of time to process the
cross-jumping optimization for the N branches in the select. The
compiler is smart enough to convert.

.HL2 VAX Compiler inefficient on select, PDP-11 is good
See example by J. Wilkerson.

.TEST PAGE 14
.LITERAL

```
from:                to:

select I from    |       if i = 1 do
case 1:          |               //....
        //....    |       orif i = 100000 do
case 100000:     |               //...
        //...     |       otherwise
default:         |               //..
        //..     |       endif
endselect        |
                 |
```

The present algorithm decides based on the time/space trade-offs on the actual number of "cases." This decision may not always be correct.

4.7 RUN TIME SUPPORT, LSI-11

4.7.1 LSI-11 Interrupt Routines

Separate the normal register interrupt routine from the floating point register interrupt routine for stand alone LSI-11s.

4.7.2 TEXTIO Separation

Separate the LSI TEXTIO into many modules so you don't have to link to them all.

4.8 BETTER DEBUGGING AND DIAGNOSTICS

4.8.1 Additional Compile-time Range Checking

Assignment of large objects (strutures or arrays) is not range checked at present. This could be done. This applies to both compile time and run time range checking.

4.8.2 Range Checking On Subscripts Of Array Constructors

4.8.3 Check Forward Procedure Same As Actual Procedure

Forward procedure statements are not completely checked to be sure that they are the same as actual procedure statement. (i.e., made actual INOUT when forward was OUT and no diagnostic.)

4.8.4 Check Size Of Actual Arg <= Size Of Formal Arg When By VAL

This may have already been done - must test.

4.8.5 Detection Of Comparisons Of Objects With Holes

An optional diagnostic to detect comparisons of objects with holes would be useful for finding code that may have had bugs with older versions of the compiler due to holes. It would also be useful to write code that runs a little faster (i.e., it's faster if holes don't need to be filled).

4.8.6 Warning On Too Few Components In Table Of Strings

Why doesnt BADTABLE.PRX warn about too few components in table? Note that if you take out blank character string, it does.

4.8.7 More Accurate Line Number Of Errors

Find line number of errors more accurately. Giving the line number of the end of the module is useless.

4.8.8 Create_Unique_Global_Symbols Compiler Directive

Add a "create_unique_global_symbols" compiler directive that causes the symbols generated in the PDP11 M11 file to be both unique within 6 characters AND have the full symbol name as follows:

```
        XXYYnn$"----------"
where
        XX = first two characters of module name
        YY = last two characters of module name
        nn = base 36 number
        ------ = identifier name string
```

4.8.9 Diagnostic When SIZEOF Result Will Not Fit.

Need a diagnostic on sizeof if size of item will not fit into size of left variable (i.e., > 2**16 bits).

4.8.10 Diagnostic Upon Export Of Dynamic Item From Main Module

A diagnostic should be issued when attempting to export a dynamic item from a main module. Apparently this is a common error made by Pascal programmers but one that Nova programmers have never had a problem with.

4.8.11 Diagnostic That Whatever Features Of Packed Packed Which Don't Work

are not implemented.

4.9 OPTIMIZATION IDEAS

4.9.1 Array Referencing On LSI-11

Array referencing into small size element arrays (i.e., 4 bits, etc.) on LSI-11 could be optimized for space.

4.9.2 Initial Conditions Of Arrays, Structures

Use P_data for storing array and structure initial conditions or constructors. As is, lists of MOV instructions are generated.

4.9.3 Register Allocation

Lots of room exists for improving register allocation. Needs study.

4.9.4 CG_Table Pattern Improvements

Some of the CG_tables are obviously less efficient than they could be.

4.9.5 Library Functions For String Operations

Library functions to do string comparisons. String descriptors - see Run Time Library users Guide, page 5-17.

4.10 VMS VERSION 4 ISSUES

4.10.1 Module Name Limit Of 9 Characters

Restriction could be removed; however, would not be backward compatible.

4.11 SUPPORT ISSUES

4.11.1 32 Bit Integer In TEXTIO

We need a 32 bit integer procedure in TEXTIO for the LSI-11. See R. Shelton's INT32.PRX.

4.11.2 Holes In The Compiler

Check all the places in the compiler that are now detected as having structures with holes. Were any other bugs solved by this fix? Can they be recoded to make the compiler run faster?

These modules have declarations with holes: OBJFIL1, OBJFIL3, PHASE3, CGOUT, CGACTION. MCODE had them, but they were removed- probable bug in reg allocation.

These modules have structure constructors with holes: ATOM, MAKCOD, LOOKUP, WALKSUB, SYNIN, MSTRC.

These modules have out val with holes: CGSPECIAL.

Note that the OPERAND structure has holes in it, and it is compared in several places in the compiler. Also, there are some declarations of it without an initially of the null_operand.

4.12 DOCUMENTATION

4.12.1 Revise/Rewrite "Programming In Praxis" Manual.

Work on this has begun. Revising all text to be accurate to the language as it exists. Removing and replacing all examples such that only those that work remain. Including useful material from the Reference Manual. Adding chapters on run time support on both the VAX and LSI-11.

4.12.2 Test Suite

Command files could be written to organize the test suite for testing purposes as well as making example programs available to the users as help files. A listing of the test suite programs would be an excellent starting point for a book of example programs.

4.12.3 Viewgraphs

We need a good set of viewgraphs as instructional aids to the language.

4.13 GENERAL SUPPORT

4.13.1 WALKER

Set up logical symbols to run WALKER. Describe its use to others. For now:

```
assign DISK$LIB:[PRAXISLIB.PRODUCTS.WALKER.VMX] MYWALK
walk :== $MYWALK:WALKER
```

4.13.2 CROSS (Phebus)

Put walker program written by Phebus into Praxis and make it work within our system.

4.13.3 User Analyzer

Move the User Analyzer program into the Praxis Product and clean it up for general use. This program analyzes all use of Praxis within specific directories and generates a chart showing the quantity use of every feature in the language.

CHAPTER 5

DETAIL CODE CHANGES TO THE COMPILER

Support for the Symbolic Debugger was a major addition to the compilers. Listed below are phrases which would indicate to someone familiar to the internals of the compiler the areas which were modified or added. There is no reason to believe that any of these changes would affect the correct operation of the compiler in compiling sources.

The changes to the common (VAX and PDP) compiler code included:

Added line_no to stack contents

Added line_no to make_atom

source_file_array

C_line_no ccode

Debug Switch on the Command Line - has no effect yet

Debug symbol table. On by default.

do_line_no_action added to CGACTION

debug_code procedure - purpose

link_debug_code procedure

line numbers loaded into lexeme

static cells for debug_code and lst_debug_code

makline procedure to insert a line number cell at end of code stream

added stkatm_line parameter to get_tree, and the line number of the atom to each call to get_tree (of which there are a great many...).

endfor_line in for statment node

endwhile_line filed in while statment node

condition line number transfered to result of zahn statment

added gmode, gdref, gcref, line_no ccodes, do_line_no action to tables

added is_variable function to Valid

add literal output of line number to all intermediate code output of Walk

new procedures OK_simple_output, OK_lit_output to support Walk

MDECL changed to set size of large unpacked structure as rounded

WALK changed to test overflow immediately after self-math instruction

VAX CGTABLES changed to use relative offsets for BVC instruction

WALKEXP changed so as not to restack variadic flexible arrays

The changes to the PDP-11 code consisted of only the addition of line number to M11 listing.

The changes to the VAX code involved major additions to the object code generation including:

add line numbers to listings of macro output

add object file debugger constant declarations

line_code_size procedure

procedure to dump the debugger block to the object file

set_line_number procedure

load_delta_pc procedure

send_term_block procedure

add_to_object procedure - must handle line numbers going backward

add_rtend_block procedure

add_line_numbers_to_object procedure

add rtn_block procedure

declare_source_files procedure - must pass list of source files

combined during this compilation to object file and debugger.

add_source_to_object procedure - adds information about the source files

pass debug information through Phase1, Phase2, Phase3 of object optimizer.